IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Thomas J. Pavela	Examiner:	Ted T. Vo
Serial No.:	To be assigned	Group Art Unit:	2122
Filed:	September 19, 2001	Docket:	ST9-98-107US2
Title:	SYSTEM AND METHOD FOR DEVELOPING TEST CASES USING A TEST OBJECT LIBRARY		

CERTIFICATE OF MAILING UNDER 37 CFR 1.10
'Express Mail' mailing label number: EL815953376US
Date of Deposit: September 19, 2001
I hereby certify that this paper or fee is being deposited with the United States Postal Service 'Express Mail Post Office To Addressee' service under 37 CFR 1.10 and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

By Isabell Ogata
Name: Isabell Ogata

PRELIMINARY AMENDMENT

Commissioner for Patents
Washington, D.C. 20231

Dear Sir:

Prior to a first Office Action, please amend the above-identified application. Amended paragraphs are presented in "clean" form with marked-up versions provided in the Appendix.

IN THE SPECIFICATION

On page 1, please replace lines 7-9, as follows:

Application Serial No. 09/222,267, entitled "GRAPHICAL USER INTERFACE FOR DEVELOPING TEST CASES USING A TEST OBJECT LIBRARY," filed on same date herewith, by Thomas J. Pavela, attorney's docket number ST9-98-108.

On page 1, please replace lines 22-28, as follows:

The past two decades have seen a rapid proliferation of computer application programs. To be competitive, software applications must be responsible to customer's rapidly evolving needs, and must be as bug-free as possible. One method of assuring

that a software release is bug free is by testing the software with a wide variety of test cases, carefully chosen to test all critical modules in the software product. Such testing procedures are especially important where the software under test is designed to operate across a number of processors or other systems operating in parallel. In such cases, the individuals writing the test cases should be familiar with the operating system, and communication protocols for all of the elements in the system, but unfortunately, there are few individuals with all of the skills necessary to write a complete test program.

On page 1, please replace lines 29 and 31, as follows:

The development and execution of software test cases also takes a large investment of time and resources, and can delay the timely introduction of a software product to the marketplace. Because software applications are written and sold in a

On page 2, please replace lines 12-22, as follows:

What is needed is an improved system and method for developing test cases. The present invention satisfies that need by offering a system that relieves the system test designer from writing the automated code from scratch. Instead, the test designer is provided with a library of test objects, each implementing a portion of the automated test procedure. When one of the objects in the library is selected, the user is prompted to select from test object options that define required test parameters, thus simplifying the process. The system and method for developing test cases also relieves the test case designer from the burden of familiarizing themselves with the protocols and interoperability requirement for each and every system element used by the software, and allows the test plan to be updated and documented with significantly less effort.

On page 4, please replace lines 9-10, as follows:

FIGs. 9A-9E are diagrams showing a listing of an automated test code generated from the script file shown in FIG. 6;

On page 10, please replace lines 3-18, as follows:

FIG. 7 is a flow chart showing an illustrative example of process steps used to generate the test plan 322 from the source file 318. In some cases, the tags in the source file 318 simply specify a print formatting (e.g. indentation and whether the test following the tag is a header, such as tags 401-450). Other tags, such as tags 602A-616A, are instead associated with the members of the library of executable test code objects 314. The first tag type can be interpreted and printed by a QPRINT function 321. However, the second tag type is not recognized by a QPRINT function 321, and must be identified and handled differently. These tags are interpreted, translated using script macros 325, and conversational language statements for the test plan 322 are generated as shown in blocks 702 and 704. This is accomplished via script macros 325 such as the script macro having representative code instructions presented in FIGs. 30A and 30B. If tags are encountered that are uninterpretable (because, for instance, the user typed in the incorrect tag name, a tag not associated with one of the script macros 325, or an unsupported or out of range tag parameter), an error message (such as "+++") noting this error is placed in the test plan 322 in the appropriate place.

On page 11, please replace lines 8-21, as follows:

After the test plan 322 has been approved, the next step is to generate test code 320 from the source file 318. This step is performed by an HPTC EXEC software module 319, which interprets the tags and associated tag parameters in the source file, and generates test code with links to the appropriate member of the library of executable code objects 314. In one embodiment, the generated automated test code 320 uses a screen-scraping technique wherein the commands to the system elements are provided via coded automated keystroke entries which are injected to the appropriate system element. An example of a portion of HPTC EXEC code used to translate a HPSRLM2 test object to test code is presented in FIG. 31. An example of the generated test code 320 translated from the HPSRLM2 test object is shown in FIG. 9E, at 910. An example of the subroutine that the test code line 910 calls is

shown in FIGs. 32A-32C. This code resides in the automated executable subroutine library 327. Responsive messages from the system element are intercepted and used to determine whether the command proceeded normally, or if there was an error.

On page 11, please replace lines 22-29, as follows:

FIGs. 9A-9E are diagrams showing a listing of an automated test code 320 generated from the script file shown in FIG. 6. The listing begins in FIG. 9A with commented code ("leading '/*' characters denote code comments) describing the objectives, and scenario for the test case. The listing continues in FIG. 9B with additional information regarding the test case name, the source file, and other information. The remainder of FIG. 9B, as well as FIGs. 9C and 9D describe variable definitions for scenario variables and called commands. FIG. 9E presents the test code commands.

On page 12, please replace lines 1-15, as follows:

Returning to FIG. 3, once the test code has been generated, it can be executed 326. Using the screen-scraping technique described above, the commands are provided to the system elements 132, and response messages from the system elements 132 are used to determine the test result, and whether the test errors were reported. If no test errors were reported 328, the test results are provided to the user, and the test case is completed. If test errors are reported 328, they may be corrected by altering the test case source filed 318, and regenerating the test plan 322 and the automated test code 320. One of the advantages provided by the present invention is the use of the source file 318 to automatically generate both the test plan 322 and the automated test code 320. When errors are detected in either the test plan 322 or the test code 320 (whether by executing the code or compiling the code), those errors can be corrected by making appropriate changes to the source file 318. Hence, the test code 320 is self-documenting, and there is no need for the user to go through the time-consuming (and often omitted) process of rewriting the test plan 322 to conform with actual test procedures.

On page 13, please replace lines 1-3, as follows:

input 210. The test case GUI 312, as described more fully herein, also permits the user to convert manually entered test case commands into test case objects, and to store them in the executable code test object library 313 for future use.

On page 14, please replace lines 13-16, as follows:

FIG. 15 is a diagram showing an example of how the test case GUI 312 prompts the test designer to enter an author name. An author field 1504 is provided in an "author name" window 1502. The test designer enters their name in the author name field 1502, and selects the "next" button 1506.

On page 17, please replace lines 9-19, as follows:

FIG. 26 is a diagram showing another embodiment of the test case GUI 312. Here, the user has selected the terminals category 2602, showing library member objects 2604 including a "use a live terminal" library member object 2606, and has further selected the "use a live terminal" library member object 2606, thus opening a "Setup a live terminal" window 2606. The "Setup a live terminal window" 2606 presents an ON frame 2612, terminal option frame 2608, a terminal frame 2614 and a port frame 2616. The required input parameters for the live terminal library member object 2606 are highlighted by changing the color of the frame or the test labeling of the frame. If the user enters a value in the combo box in the terminal option frame 2608, that value is saved for the duration of the test case development session. Tooltip box 2610 is applied to the ON frame 2612 to provide the user with context sensitive help.

IN THE DRAWINGS

Proposed Drawing Changes is attached indicating a typographical error as follows. "FIG. 9F" should read –FIG. 9E--.

IN THE CLAIMS

Please cancel claims 1-21 and add new claims 22-36 as follows:

22. (NEW) A method of generating test code for an automated test procedure applyable to a system comprising a plurality of interconnected elements, the method comprising the steps of:

defining a source file having a plurality of tags, each tag associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure;

generating a test plan in a conversational language from the source file; and

generating the test code for the automated test procedure from the source file.

23. (NEW) The method of claim 22, wherein the step of generating a test plan comprises the steps of:

translating the tags; and

generating a conversational language phrase for each translated tag.

24. (NEW) The method of claim 23, wherein the test plan comprises a test index identifying the system elements tested by the test code, the test index generated by performing the step of scanning the interpreted tags to identify the system elements tested by the test code.

25. (NEW) The method of claim 23, wherein the step of generating a test plan further comprises the steps of:

identifying an uninterpretable tag in the test plan; and

appending the test plan with an error message identifying the uninterpretable tag.

26. (NEW) The method of claim 22, wherein the step of generating test code for the automated test procedure comprises the step of translating the executable code objects associated with the tag in the source file.

27. (NEW) An apparatus for generating test code for an automated test procedure applyable to a system comprising a plurality of interconnected elements, comprising:

means for defining a source file having a plurality of tags, each tag associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure;

means for generating a test plan in a conversational language from the source file; and

means for generating the test code for the automated test procedure from the source file.

28. (NEW) The apparatus of claim 27, wherein the means for generating a test plan comprises:

means for translating the tags; and

means for generating a conversational language phrase for each translated tag.

29. (NEW) The apparatus of claim 28, wherein the test plan comprises a test index identifying the system elements tested by the test code, wherein the test index generated by performing the step of scanning the interpreted tags to identify the system elements tested by the test code.

30. (NEW) The apparatus of claim 28, wherein the means for generating a test plan further comprises:

means for identifying an uninterpretable tag in the test plan; and

means for appending the test plan with an error message identifying the uninterpretable tag.

31. (NEW) The apparatus of claim 27, wherein the means for generating test code for the automated test procedure comprises means for translating the executable code objects associated with the tag in the source file.

32. (NEW) A program storage device, readable by a computer, tangibly embodying at least one program of instructions executable by the computer to perform method steps of generating test code for an automated test procedure applyable to a system comprising a plurality of interconnected elements, the method comprising the steps of:

defining a source file having a plurality of tags, each tag associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure;

generating a test plan in a conversational language from the source file; and

generating the test code for the automated test procedure from the source file.

33. (NEW) The program storage device of claim 32, wherein the method step of generating a test plan comprises the method steps of :

translating the tags; and

generating a conversational language phrase for each translated tag.

34. (NEW) The program storage device of claim 33, wherein the test plan comprises a test index identifying the system elements tested by the test code, the test index generated by performing the step of scanning the interpreted tags to identify the system elements tested by the test code.

35. (NEW) The program storage device of claim 33, wherein the step of generating a test plan further comprises the method steps of:

identifying an uninterpretable tag in the test plan; and

appending the test plan with an error message identifying the uninterpretable tag.

36. (NEW) The program storage device of claim 32, wherein the method step of generating test code for the automated test procedure comprises the method step of translating the executable code objects associated with the tag in the source file.

REMARKS

Prior to a first Office Action in this application, Applicant requests that original claims 1-21 be cancelled and new claims 22-36 be added. The new claims do not involve any new matter or objectionable changes. When the Examiner takes this application up for action, he is requested to take the foregoing into account.

I. THE APPLICANT'S INVENTION IS PATENTABLE OVER THE DELONG REFERENCE

A. The DeLong Reference

U.S. Patent No. 5,892,947, issued April 6, 1999 to DeLong et al. discloses a test support tool system and method. A test support tool system and method produce software test programs from a logical description of selected software. Test programs are created by producing a cause-effect graph from the logical description, creating a decision table, producing test cases, and synthesizing test cases into a test program. The test support tool system includes an interface for receiving a logical description of software, a logical database, an analysis and test case generation module, a control module, and a test program synthesis module.

B. The Subject Invention

The Applicant's invention is described by a method, apparatus, article of manufacture, and a memory structure for generating a test code for an automatic procedure is disclosed. The method comprises the steps of defining a source file having a plurality of tags associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure, generating a test plan in a conventional language from the source file, and generating an automated test code for the automated test procedure from the source file. In one embodiment, a test index identifying system elements tested by the test code is generated and incorporated into the test plan, allowing the user to verify that all desired system elements are exercised by the automated test code. The article of manufacture comprises a data storage device tangibly embodying instructions to perform the method steps described above.

The apparatus comprises means for defining a source file having a plurality of tags, wherein each tag is associated with a member of a library of executable code objects defining a set of instructions for performing a portion of an automatic test procedure, means for generating a test plan in a conversational language from the source file, and means for generating an automated test code for the automatic test procedure from the source file.

C. Patentable Differences Between the DeLong Reference and the Applicant's Invention

With Respect to Claims 22, 23, 27, 28, 32, and 33: The DeLong reference fails to teach the step of "defining a source file having a plurality of tags associated with a member of a library of executable code objects defining a set of instructions for performing a portion of the automatic test procedure," as described in claims 22 and 23. Claims 27, 28, 32, and 33 include similar limitations.

The DeLong reference appears to disclose the use of a "logical description" which has "tags", but does not disclose that the tags are associated with executable code objects defining an instruction set for performing a portion of the automatic test procedure. Indeed, the tags disclosed in the DeLong reference appear to be for annotation purposes only. Specifically, the DeLong reference discloses:

> "Logical description 22 according to one embodiment has elements including a modification history of the logical description; including references, documents or other sources, referred to in preparing logical description 22; including tags, i.e., specific items in the references that are addressed by this logical description; including primitive conditions, i.e., atomic formulae expressing candidate conditions on the input domain of the function being described; including constraints, i.e., additional facts that express relations on the input conditions, on the logical description, or on test cases generated from a logical description; including operation, i.e., signature (number and type of arguments and results) of the function being described and auxiliary information needed to generate concrete test cases; and including effects, i.e., clauses or well-formed formulae constituting a pure logic program for the input/output relation of the function being described." (col. 4, line 25-52)

The fact that the DeLong reference does not refer to tags that are associated with a member of a library of executable code objects is further evidenced by the fact that nothing in

the DeLong reference discloses the use of the tags to generate conversational language phrases for the translated tags, as is required by claim 23.

With Respect to Claim 24, 29 and 34: Claim 24 specifies that:

> *"the test plan comprises a test index identifying the system elements tested by the test code, the test index generated by performing the step of scanning the interpreted tags to identify the system elements tested by the test code."*

The Applicant can find no disclosure of an index identifying the system elements tested by the test code, nor an index generated by scanning the interpreted tags. That these elements are missing from the DeLong reference is further evidence that the "tags" referred to above are not analogous to the tags of the Applicant's invention.

Claims 29 and 34 include analogous limitations and are patentable on this basis.

D. Dependent Claims

Claims 25, 26, 30, 31, 36, and 36 include the limitations of the claims dependent thereon and are patentable on this basis alone. Further, claims 25, 26, 30, 31, 36, and 36 include further limitations rendering them even more remote from the prior art known to the Applicant. Accordingly, these claims are allowable as well.

II. CONCLUSION

It is submitted that this application is now in good order for allowance and such allowance is respectively solicited. Should the Examiner believe minor matters still remain that can be resolved in a telephone interview, the Examiner is urged to call Applicant's undersigned attorney.

Respectfully submitted,

Thomas J. Pavela

By his attorneys,

GATES & COOPER LLP

Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, California 90045
(310) 641-8797

Date: September 19, 2001

By: [signature]
Name: Victor G. Cooper
Reg. No.: 39,641

APPENDIX : SPECIFICATION IN MARKED-UP FORM

On page 1, please replace lines 7-9, as follows:

Application Serial No. [--/---,---] 09/222,267, entitled "GRAPHICAL USER INTERFACE FOR DEVELOPING TEST CASES USING A TEST OBJECT LIBRARY," filed on same date herewith, by Thomas J. Pavela, attorney's docket number ST9-98-108.

On page 1, please replace lines 22-28, as follows:

The past two decades have seen a rapid proliferation of computer application programs. To be competitive, software applications must be responsible to customer's rapidly evolving needs, and must be as bug-free as possible. One method of assuring that a software release is bug free is by testing the software with a wide variety of test cases, carefully chosen to test all critical modules in the software product. Such testing procedures are especially important where the software under test is designed to operate across a number of processors or other systems operating in parallel. In such cases, the individuals writing the test cases should be familiar with the operating system, and communication protocols for all of the elements in the system, but unfortunately, there are few individuals with all of the skills necessary to write a complete test program.

On page 1, please replace lines 29 and 31, as follows:

The development and execution of software test cases also takes [an] a large investment of time and resources, and can delay the timely introduction of a software product to the marketplace. Because software applications are written and sold in a

On page 2, please replace lines 12-22, as follows:

What is needed is an improved system and method for developing test cases. The present invention satisfies that need by offering a system that relieves the system test designer from writing the automated code from scratch. Instead, the test designer is provided with a library of test objects, each implementing a portion of the

automated test procedure. When one of the objects in the library [are] is selected, the user is prompted to select from test object options that define required test parameters, thus simplifying the process. The system and method for developing test cases also relieves the test case designer from the burden of familiarizing themselves with the protocols and interoperability requirement for each and every system element used by the software, and allows the test plan to be updated and documented with significantly less effort.

On page 4, please replace lines 9-10, as follows:

FIGs. 9A-9[F]E are diagrams showing a listing of an automated test code generated from the script file shown in FIG. 6;

On page 10, please replace lines 3-18, as follows:

FIG. 7 is a flow chart showing an illustrative example of process steps used to generate the test plan 322 from the source file 318. In some cases, the tags in the source file 318 simply specify a print formatting (e.g. indentation and whether the test following the tag is a header, such as tags 401-450). Other tags, such as tags 602A-616A, are instead associated with the members of the library of executable test code objects 314. The first tag type can be interpreted and printed by a QPRINT function 322. However, the second tag type [are] is not recognized by a QPRINT function 321, and must be identified and handled differently. These tags are interpreted, translated using script macros 325, and conversational language statements [in] for the test plan 322 are generated as shown in blocks 702 and 704. This is accomplished via script macros 325 such as the script macro having representative code instructions presented in FIGs. 30A and 30B. If tags are encountered that are uninterpretable (because, for instance, the user typed in the incorrect tag name, a tag not associated with one of the script macros 325, or an unsupported or out of range tag parameter), an error message (such as "+++") noting this error is placed in the test plan 322 in the appropriate place.

On page 11, please replace lines 8-21, as follows:

After the test plan 322 has been approved, the next step is to generate test code 320 from the source file 318. This step is performed by an HPTC EXEC software module 319, which interprets the tags and associated tag parameters in the source file, and generates test code with links to the appropriate member of the library of executable code objects 314. In one embodiment, the generated automated test code 320 uses a screen-scraping technique wherein the commands to the system elements are provided via coded automated keystroke entries which are injected to the appropriate system element. An example of a portion of HPTC EXEC code used to translate a HPSRLM2 test object to test code is presented in FIG. 31. An example of the generated test code 320 translated from the HPSRLM2 test object is shown in FIG[s]. 9[F]E, at 910. An example of the subroutine that the test code line 910 calls is shown in FIGs. 32A-32C. This code resides in the automated executable subroutine library 327. Responsive messages from the system element are intercepted and used to determine whether the command proceeded normally, or if there was an error.

On page 11, please replace lines 22-29, as follows:

FIGs. 9A-9[F]E are diagrams showing a listing of an automated test code 320 generated from the script file shown in FIG. 6. The listing begins in FIG. 9A with commented code ("leading '/*' characters denote code comments) describing the objectives, and scenario for the test case. The listing continues in FIG. 9B with additional information regarding the test case name, the source file, and other information. The remainder of FIG. 9B, as well as FIGs. 9C and 9D describe variable definitions for scenario variables and called commands. FIG. 9[F]E presents the test code commands.

On page 12, please replace lines 1-15, as follows:

Returning to FIG. 3, once the test code has been generated, it can be executed 327. Using the screen-scraping technique described above, the commands are provided to the system elements 132, and response messages from the system

elements 132 are used to determine the test result, and whether the test errors were reported. If no test errors were reported 328, the test results are provided to the user, and the test case [has] is completed. If test errors are reported 328, they may be corrected by altering the test case source filed 318, and regenerating the test plan 322 and the automated test code 320. One of the advantages provided by the present invention is the use of the source file 318 to automatically generate both the test plan 322 and the automated test code 320. When errors are detected in either the test plan 322 or the test code 320 (whether by executing the code or compiling the code), those errors can be corrected by making appropriate changes to the source file 318. Hence, the test code 320 is self-documenting, and there is no need for the user to go through the time-consuming (and often omitted) process of rewriting the test plan 322 to conform with actual test procedures.

On page 13, please replace lines 1-3, as follows:

input 210. The test case GUI 312, as described [for] more fully herein, also permits the user to convert manually entered test case commands into test case objects, and to store them in the executable code test object library 313 for future use.

On page 14, please replace lines 13-16, as follows:

FIG. 15 is a diagram showing an example of how the test case GUI 312 prompts the test designer to enter an author name. [A] An author field 1504 is provided in an "author name" window 1502. The test designer enters their name in the author name field 1502, and selects the "next" button 1506.

On page 17, please replace lines 9-19, as follows:

FIG. 26 is a diagram showing another embodiment of the test case GUI 312. Here, the user has selected the terminals category 2602, showing library member objects 2604 including a "use a live terminal" library member object 2606, and has further selected the "use a live terminal" library member object 2606, thus opening a "Setup a live terminal" window 2606. The "Setup a live terminal window" 2606

presents an ON frame 2612, terminal option frame 2608, a terminal frame 2614 and a port frame 2616. The required input parameters for the live terminal library member object 2606 are highlighted by changing the color of the frame or the test labeling of the frame. If the user enters a value in the combo box in the terminal option frame 2608, that value is saved for the duration of the test case development session. Tooltip box 2610 is applied to the ON frame 2612 to provide the user with context sensitive help.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Thomas J. Pavela	Examiner:	Ted T. Vo
Serial No.:	To be assigned	Group Art Unit:	2122
Filed:	September 19, 2001	Docket:	ST9-98-107US2
Title:	SYSTEM AND METHOD FOR DEVELOPING TEST CASES USING A TEST OBJECT LIBRARY		

CERTIFICATE OF MAILING UNDER 37 CFR 1.10
'Express Mail' mailing label number: EL815953376US
Date of Deposit: September 19, 2001
I hereby certify that this paper or fee is being deposited with the United States Postal Service 'Express Mail Post Office To Addressee' service under 37 CFR 1.10 and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

By: Isabell Ogata
Name: Isabell Ogata

PROPOSED DRAWING CHANGES

Commissioner for Patents
Washington, D.C. 20231

Dear Sir:

Applicant herewith submits a proposed drawing change to FIG. 9F in the above-identified application as follows:

"FIG. 9F" should read –FIG. 9E--.

Two copies of the drawing are enclosed wherein one copy indicates the drawing changes in red and the other copy is a proposed formal version.

Respectfully submitted,

Thomas J. Pavela

By his attorneys

GATES & COOPER LLP

Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, CA 90045
(310) 641-8797

Date: September 19, 2001

By: Victor G. Cooper
Name: Victor G. Cooper
Reg. No.: 39,641

```
/*=========================begin test case=============================*/
/*------------------------------>>> EC1 <<<-----------------------------*/
Call SwitchEC "EC1"
CONFIGURATION=30
RESTART_VTAM="YES"
ARCDEFLT="YES"
ARM="NO"
Call Hpcs_entry ""
/* load the database(s) using sharelevel 3                              */
DATABASES=" DJK "
ShareDB="YES"
Call Hpcs_load_databases "3  "
910 Call Hpcs_Start_IRLMs_21 ""
/************************************************************************/
/* Cold start IMS TM_DB region on ALL system(s)                         */
/* CQS will be started and the default model is SMQ$C19X.               */
/* The following IMS parms will be used if they are not set by the      */
/* user in IMSPARMS:                                                    */
/*   IRLM=Y,VSPEC=HP,IMSID=IMSx                                         */
/*   SHAREDQ=%%x,DC=C0x                                                 */
/*     note: x is 1,2,or 3 depending on which CEC                       */
/*   DLINM=HPC%CSA% (if DBDLIST or PSBLIST is specified in HPENTRY)     */
/************************************************************************/
CFNAMES1='CFNAMES,CFIRLM=LT01,CFVSAM=,CFOSAM=OSAMSESXI'
CFNAMES2="NO"
DATABASES=" DJK "
SHARER="NO"
HYPER="NO"
IMSLOCAL="N"
RESLIB="C"
PROCNAME="DEFAULT"
PARM1=""
PARM2=""
VSPEC="DEFAULT"
MODEL="DEFAULT"
Call Start_IMS_on_all_systems
Call Start_Tran_Scenario_1 "LEAVE=NO NTRANS=1000 ON=ALL STARTAPL=ALL"
Call Start_Tran_Scenario_1 "LEAVE=NO NTRANS=500 ON=ALL STARTAPL=ALL"
    Call Stop_all_IMSs "   "
Call Hpcs_exit ""
/*=========================end test case=============================*/
EXIT 0
INCLUDE "HPC$SUB"
/*====================== HPTC Translation summary ==================*/
/* Number of lines written = ...176                                   */
/* Number of +++ errors    = .....0                                   */
/*====================== End  Translation summary ==================*/
```

FIG. 9E

```
/*=======================begin test case========================== */
/*-------------------------------------->>> EC1 <<<-------------------------------------- */
Call SwitchEC "EC1"
CONFIGURATION=30
RESTART_VTAM="YES"
ARCDEFLT="YES"
ARM="NO"
Call Hpcs_entry " "
/* load the database(s) using sharelevel 3                                              */
DATABASES=" DJK "
ShareDB="YES"
Call Hpcs_load_databases "3  "                                          910
Call Hpcs_Start_IRLMs_21 " "
/*******************************************************************************************/
/* Cold start IMS TM_DB region on ALL system(s)                                         */
/* CQS will be started and the default model is SMQ$C19X.                               */
/* The following IMS parms will be used if they are not set by the                      */
/* user in IMSPARMS:                                                                    */
/*       IRLM=Y, VSPEC=HP, IMSID=IMSx                                                   */
/*       SHAREDQ=%%x, DC=COx                                                            */
/*              note: x is 1,2, or 3 depending on which CEC                             */
/*       DLINM=HPC%CSA%  (if DBDLIST or PSBLIST is specified in HPENTRY)                */
/*******************************************************************************************/
CFNAMES1= 'CFNAMES, CFIRLM=LT01, CFVSAM=, CFOSAM=OSAMSESXI'
CFNAMES2="NO"
DATABASES=" DJK "
SHARER="NO"
HYPER="NO"
IMSLOCAL="N"
RESLIB="C"
PROCNAME="DEFAULT"
PARM1=" "
PARM2=" "
VSPEC="DEFAULT"
MODEL="DEFAULT"
Call Start_IMS_on_all_systems
Call Start_Tran_Scenario_1 "LEAVE=NO NTRANS=1000 ON=ALL STARTAPL=ALL"
Call Start_Tran_Scenario_1 "LEAVE=NO NTRANS=500 ON=ALL STARTAPL=ALL"
      Call Stop_all_IMSs "     "
Call Hpcs_exit " "
/*=======================end test case============================ */
EXIT 0
INCLUDE "HPC$SUB"
/*==================HPTC Translation summary=================== */
/* Number of lines written  = ...176                                                    */
/* Number of +++ errors     = .....0                                                    */
/*==================End Translation summary=================== */
```

FIG. 9E